

Node.js + MongoDB Workshop

What is Node.js?

- async i/o for V8 javascript
- single threaded
- callback based
- built in libraries
- npm

What is MongoDB?

- document database
 - json
 - bson

- replication
 - high availability

- sharding

- no multi-document transactions
- no joins
- document level \$atomic operators

Hands on !

1) Write a Hello World web server in Node.js

ex1.js

2) Connect to MongoDB

Install the mongodb driver for node.js

npm install mongodb

Connect to mongodb, insert a document, find the document and update the document.

ex2.js

Exercise:

\$set a new property of the document
print the contents of all documents in the collection

Replica Sets

“Database replication ensures redundancy, backup, and automatic failover. Replication occurs through groups of servers known as replica sets.”

<http://docs.mongodb.org/manual/replication/>

Hands on!

Setting up a test replica set:

Create directories for each mongod instance:

```
mkdir /data/rs0 /data/rs1 /data/rs2
```

Start three instances of mongod:

```
mongod --dbpath /data/rs0 --port 27017 --replSet myset  
mongod --dbpath /data/rs1 --port 27018 --replSet myset  
mongod --dbpath /data/rs2 --port 27019 --replSet myset
```

You probably want to also pass the following three flags which help keep the size of your test replica members small.

```
--smallfiles --noprealloc --oplogSize 5
```

Connect to the any mongod instance:

```
mongo --port 27018
```

Initialize the replica set:

```
rs.initiate()
```

After a few seconds the mongod process you are connected to will become the replica set primary. The prompt will change to "PRIMARY>".

Run `rs.status()` to see the current configuration:

```
PRIMARY> rs.status()

{ "set" : "myset",
  "date" : ISODate("2012-05-11T17:56:37Z"),
  "myState" : 1,
  "members" : [
    { "_id" : 0,
      "name" : "<your hostname>:27018",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "optime" : { "t" : 1336758831000, "i" : 1 },
      "optimeDate" : ISODate("2012-05-11T17:53:51Z"),
      "self" : true }],
  "ok" : 1
}
```

Add the other members:

```
PRIMARY> rs.add('<your hostname>:27017')
PRIMARY> rs.add('<your hostname>:27019')
```

Take down the master.

```
db.runCommand({ replSetStepDown: 1});
```

Observe an election take place. Now bring the node back up.

More hands on!

Go back to our `node.js / mongodb` script and change the URI to point to the running replica set. Validate it still works.

`ex3.js`

Node.js / MongoDB ecosystem

<https://npmjs.org/>

Most Starred

- express
- request
- async
- socket.io
- mocha
- ...
- mongoose

<http://plugins.mongoosejs.com/>

Mongoose

<http://mongoosejs.com/>

Schema driven

Schema enforcement with MongoDB happens in the application layer
(not in MongoDB)
JavaScript is not strongly typed

Schemas compile into Models.

Models generate + fetch Documents.

Hands On!

Code along while we build a tiny mongoose app.

ex4.js

<http://mongoosejs.com/docs/schematypes.html>

- Design a schema
 - name (required)
 - attributes
 - created (add a default)
 - Generate the model
 - Create a document
 - Save a document

 - Open the mongodb shell
 - Find the document (*notice the collection name may be pluralized*)

 - Find the document using mongoose
 - Enable debugging
 - Modify the document + save
 - Inspect the compiled update clause
- <http://docs.mongodb.org/manual/core/update/#update-operators>

Adding Document methods

<http://mongoosejs.com/docs/guide.html#methods>

We can add custom methods to our document instances by defining them in our schema using its `method` method.

```
schema.method('name', function);
```

Hands on!

Let's add a method to our product schema to allow us to add the product to an Order.

```
ex4a.js
```

Adding Model static methods

<http://mongoosejs.com/docs/guide.html#statics>

Custom static methods can be added to our Models by declaring them on our schema using its `static` method.

```
schema.static('name', function);
```

Hands on!

Let's add a static Model method to help us look up all products in a given order.

ex4b.js

Middleware

<http://mongoosejs.com/docs/middleware.html>

Middleware are functions which are passed control of flow during execution of ``init``, ``validate``, ``save``, and ``remove`` methods.

Middleware is useful for atomizing model logic and repeated tasks. Say we wanted to track when the document was last updated. Middleware makes this simple.

There are two types of middleware: “pre” and “post”.

Pre:

- receives flow control
- executes before the invoked method

Post:

- does not receive flow control
- executes after all middleware and the method itself

Hands On!

Add a new Date property called “updated” to our schema. This property must be updated every time we save the document. We’ll use middleware to make sure the property gets updated each time we execute `save()`.

ex5.js

Async middleware

If a pre middleware needs to execute asynchronously, we can call the ``next`` argument later, which allows us to notify mongoose that our task is complete. Passing an Error to ``next`` will interrupt the flow and pass the error to the users callback.

Hands On!

Modify our middleware to execute asynchronously and pass an error along using ``next()``. Notice how the error is returned to the users callback.

ex6.js

Plugins

<http://mongoosejs.com/docs/plugins.html>

Schemas are pluggable, that is, they allow for applying pre-packaged capabilities to extend their functionality. Adding paths, virtual fields, middleware, model methods and statics are all supported.

A plugin is a function that accepts a schema and an optional `options` argument.

```
function neato (schema, options) {  
  schema.add({ path: type });  
  if (options && options.whatever) {  
    doOtherStuff(schema);  
  }  
}
```

To use a plugin, call the `plugin` method of the schema to which we are applying the plugin.

```
mySchema.plugin(neato, { whatever: true })
```

Hands on!

Let's convert our last updated logic into a plugin.

ex7

Community plugins:

A big benefit of plugins is not only in reusing functionality within your own projects but also in publishing them to npm for others to use. Plus, we get to be lazy and make use of functionality developed by others!

<http://plugins.mongoosejs.com/>

updates once a day

Hands on!

Let's convert our plugin module to something that we can publish to npm.

```
npm publish  
npm show "name"  
npm unpublish
```

Exercise: Modeling products + comments

In this group exercise, we're going to take what we've learned about MongoDB and try to come up with a basic but reasonable data model for an e-commerce site. For users of RDBMSs, the most challenging part of the exercise will be figuring out how to construct a data model when joins aren't allowed. We're going to attempt to model for the following entities and features:

1. Products. The core of an e-commerce site, products vary quite a bit. In addition to the standard production attributes, we'll want to allow for variations of product type, along with custom attributes.
2. Product pricing: current prices as well as price histories.
3. Product categories. Every e-commerce site includes a category hierarchy. We need to allow for that hierarchy, and we also must persist the many-to-many relationship between products and categories.
4. Product reviews. Every product has zero or more reviews, and each review can receive votes and comments.

Population

<http://mongoosejs.com/docs/populate.html>

There are no joins in MongoDB but sometimes we still want references to documents in other collections. This is where population comes in. Population is the process of automatically replacing specified paths in the document with document(s) from other collection(s).

```
Model.findById(id).populate('order comments').exec(callback)
```

Additional Resources

- <http://mongoosejs.com>
- <http://plugins.mongoosejs.com>
- <http://mongodb.github.io/node-mongodb-native/>
- <http://www.mongodb.org/>
- <https://npmjs.org/> - most starred
- <http://expressjs.com/>
- <http://jade-lang.com/>
- <https://developer.mozilla.org/en-US/docs/JavaScript>
- <http://docs.mongodb.org/manual/reference/operators/>
- <http://docs.mongodb.org/manual/core/update/#update-operators>
- <http://docs.mongodb.org/manual/aggregation/>
- <http://visionmedia.github.io/mocha/>
- <https://github.com/caolan/async>
- <http://docs.mongodb.org/manual/tutorial/control-access-to-mongodb-with-authentication/>
- <http://docs.mongodb.org/manual/sharding/>
- http://mongoosejs.com/docs/api.html#query_Query-stream
- <https://github.com/LearnBoost/express-mongoose>
- <https://github.com/visionmedia/n>
- <https://github.com/aheckmann/m>
- <https://education.10gen.com>
- <http://www.10gen.com/products/mongodb-subscriptions>

Google Groups

- mongodb: <http://groups.google.com/group/mongodb-user>
- mongoose: <http://groups.google.com/group/mongoose-orm>
- node-mongodb-native: <http://groups.google.com/group/node-mongodb-native>
- mongodb-announce: <http://groups.google.com/group/mongodb-announce>
- node.js: <http://groups.google.com/group/nodejs>

Additional Resources continued

Twitter

- @mongoosejs
- @mongodb
- @nodejs
- @npmjs

Books

- <http://www.10gen.com/books>
- <http://smashingnode.com/>